Lessons Learned From Developing A Streaming Data Framework for Scientific Analysis*

Kevin R. Wheeler
Computational Sciences Division
NASA Ames Research Center
MS 269-1 Moffett Field, CA, 94035
Kevin.R.Wheeler@nasa.gov
Mark Allan, Charles Curry
QSS Group Inc.,
MS 269-1 Moffett Field, CA 94035

Abstract

We describe the development and usage of a streaming data analysis software framework. The framework is used for three different applications: Earth science hyper-spectral image analysis, Electromyograph pattern detection, and Electroencephalogram state determination. In each application the framework was used to answer a series of science questions which evolved with each subsequent answer. This evolution is summarized in the form of lessons learned.

1 Introduction

Applying machine learning research to real-world science problems necessitates the melding of machine learning with the physical science domain. In working with Earth and life scientists it has become clear that there is never enough time, despite the interest, to become expert in the many facets of machine-based pattern recognition as well as in a science domain such as atmospheric physics or neuroscience. There is an abundance of pattern recognition code (in C, Fortran, Matlab, S-Plus all in evolutionary flux) available for scientists to apply; unfortunately, each algorithm has its own pitfalls and requires that the data be in a particular format. In fact, the two areas in which we have spent the most time are in understanding the scientific problem and in reformatting and manipulating the data.

The issue of what question to ask for a particular data oriented problem forms the underpinnings of science and is well beyond the scope of this paper. However, it does make sense to understand which questions are possible to be answered given the current state of machine learning and the methods of data collection. We have observed that scientists can be swayed to ask the wrong question in an attempt to use an available algorithm to derive an answer. A common example of this is to ask what variables are most contributing to the observed variance via application of Principal Component Analysis. This assumes that the underlying data generation is best represented by a Gaussian process. A better question but much more difficult to answer would be to ask how to create a parameterizable model from which we can infer the meaning behind the observed variance. It is only through a close collaboration between machine learning researchers and the domain experts that it is possible to overcome this difficulty. Once the appropriate question has been asked, it should be possible to see if any tools are available to provide a relevant answer. The issue now becomes one of algorithmic availability, knowing that the code will be used by an over-committed scientist who does not have time to debug others' code, to reformat his data to fit into a new file format, or time to code the latest algorithms from scratch.

In our work with Earth and life scientists we have developed a software framework that provides a graphical depiction of the algorithmic steps and allows for the scientists to manipulate the data and algorithm during data streaming. This Signal Processing Environment for Algorithmic Development (SPEAD) has been applied to hyper-spectral data understanding as well as pattern recognition for Electromyographic (EMG) and Electroencephalographic (EEG) data. In this paper we present lessons learned from the perspective of machine learning researchers in their efforts to help both Earth and life scientists with their data domain problems. We will discuss results in both fields concerning the particular recognition problems.

The Earth science problem consisted of measuring solar flux with 384 spectral bands from 380nm to 2000nm. The goal of the data analysis was to determine what components in the atmosphere were most affecting the radiative energy

*Supported by NASA Intelligent Systems/IDU program.

transfer from the Sun to Earth [1]. In the life science project, we were interested in measuring/monitoring both EMG and EEG signals from participants that were performing control tasks. The goal was to create pattern recognition algorithms which would map the EMG [2] and EEG [3] signals to real-time robotic control commands. Although both problem domains were extremely different, the framework addresses the same underlying issues and thus has proved to be helpful in both.

In the following sections the system requirements of the developed framework are described. This is followed by details on the framework implementation and then the three applications are discussed and the lessons learned are summarized.

2 System Requirements Analysis

When we first started considering algorithm development for Earth and life scientists we looked for a solution which would allow a user to create new pattern recognition algorithms that would be easy for the domain scientist to manipulate and at the same time powerful enough to accomplish significant tasks. We also wanted a system which was portable (Linux, Mac OS X, IRIX), would run on single and multiple machines, had very fast interactive 3-D graphics, and would support software packages from public to commercial domains (such as Matlab and differential equation solvers). One of the key successes to our development efforts was to focus on providing a smooth transition between initial batch oriented static analysis to streaming applications for use during live data collection. After comparing several available packages out of the huge number available, we decided to start from scratch while working with scientists in their problem domains. We considered the Aurora package from Brown University [4], Sapphire from Lawrence Livermore National Laboratory [5], ADaM from the University of Alabama at Huntsville, Advanced Visual Systems products (AVS), Simulink (The Mathworks), and our own package SPEAD.

In particular we look at the form of the application (scripts, or GUI wiring diagrams), the form of the network nodes, the method of interconnecting the nodes, whether it is optimized for batch or streaming data, and the visualization capabilities. Each package has its strengths to fulfill a particular niche. Aurora works directly with databases, Sapphire works with databases and has C++ nodes accessed via Python scripts, Simulink is for streaming control work, and AVS has extensive 3-D data visualization. ADaM has facilities to interface directly with databases and has extensive image processing capabilities to work with standard formats such as HDF, however it is more focused on batch applications than on supporting streaming. None of the packages seemed to fulfill most of the requirements that we wanted in a single package.

To achieve an easy-to-program interface, the front end of the system is graphical, whereby programs are created using wiring diagrams representative of a data flow methodology. We did not want to make the graphical front end a requirement for using the processing routines and therefore the front end is used to generate standard Unix scripts that call the standalone processing routines. These scripts can be modified after generation without using the graphics or can be written from scratch.

Another requirement of the system is that we have the ability to run the wiring network across multiple heterogeneous machines by specifying which part of the network should run on which machines. Our main goal was to keep everything as simple as possible, including the mechanism for passing data from one machine to the next. Therefore, we chose to use socket code and shared memory as our means of communicating from machine to machine and between processes within a single machine respectively. The specifics of these implementations are hidden from the users through the use of wiring diagrams.

From the outset of the development it was clear that the scientists wanted one system which would perform in two very different modes. At the beginning of data exploration, it was very common for scientists to use batch oriented file based methods to analyze their data. Once a working proof-of-concept algorithm had been developed, scientists turned to using this algorithm on live streams of data. Clearly most of batch methods are not immediately usable for stream analysis. However, many times the transition is difficult due to computing infrastructure problems (i.e. how to get the data from here to ‘there’ where ‘there’ may be multiple machines) rather than creativity of solving the problem. Two approaches were followed in network creation: a batch process was used to analyze the data and then windowed to use in a streaming application, or the algorithm started as a streaming method. The beauty of using the framework to answer static questions is that the network that results in the solution can then be easily used in an on-line method. In particular, if we establish a network that improves its estimates with each successive data point, this will then require no modification when transitioning from the static to the dynamic case.

Fast interactive graphics were essential for our application. The graphics needed to be fast enough to render large sets as they flowed by and also to provide the capability to manipulate the plots as the data was changing (i.e. zoom, rotate, translate in 2-D and 3-D, etc.). We chose to code all of the graphics using C++ with OpenGL.

One of the most useful capabilities that we have incorporated into the system is the ability to easily swap algorithm components. This allows scientists to apply different ideas and to modify their approaches immediately upon seeing the results.

To avoid reinventing the wheel for existing routines which work only in their native environments, we have a shared memory interface with The Mathwork's Matlab software. This links the variable space of our environment with Matlab's variable space.

Throughout years of programming signal processing systems, our experience has been that the most effective language is one that requires the least memorization of complicated syntax. In particular, those systems that provide graphical syntax feedback during programming are less demanding in this respect. This allows for novice users to accomplish more complicated tasks expediently. The expert programming community will certainly want the power and flexibility to program at a low level to achieve very specific tasks. We are not proposing to change this methodology; we want to provide greater capabilities to those users who do not have time to become experts.

3 Wiring Diagram Example

The data flow wiring diagram user interface has been successfully employed for years both in the commercial sector (such as Labview from National Instruments, and Simulink from The Mathworks) and also in academia (such as Khoros Cantata from the University of New Mexico). With appropriate graphical feedback, this type of interface allows for novices to program rather complicated systems, and for experts to design new algorithms. We were inspired both by Cantata and Labview in that Cantata had each routine as a separate executable (and function call), and Labview allows for complicated data structures and for "for" loops. In our system we decided to separate the flow of the data from the control and parameter values. Visually, parameter flow is from left to right (inputs/outputs), while data flow is top to bottom (reads/writes).

Variable data types are color encoded for type to let the user know which inputs/outputs may be connected. Also the user is prevented from connecting inappropriate types together during the wiring stage. Explicit casting of types is possible if the user wants to override this behavior. Units within a network may be grouped together to aid in depiction and separation of the algorithm. An example of a network that implements a particle filter [6] for continuous state tracking is shown in Figure 1. On the far left hand side of this figure is a list of categories of each of the types of available units. In this example, data is streamed in and the particles are used to track how a specified model should be scaled upon sequential presentation of each data point. The magnified view of the filter loop is also presented in Figure 1 depicts how easy it is to create a continuous feedback mechanism within a network.

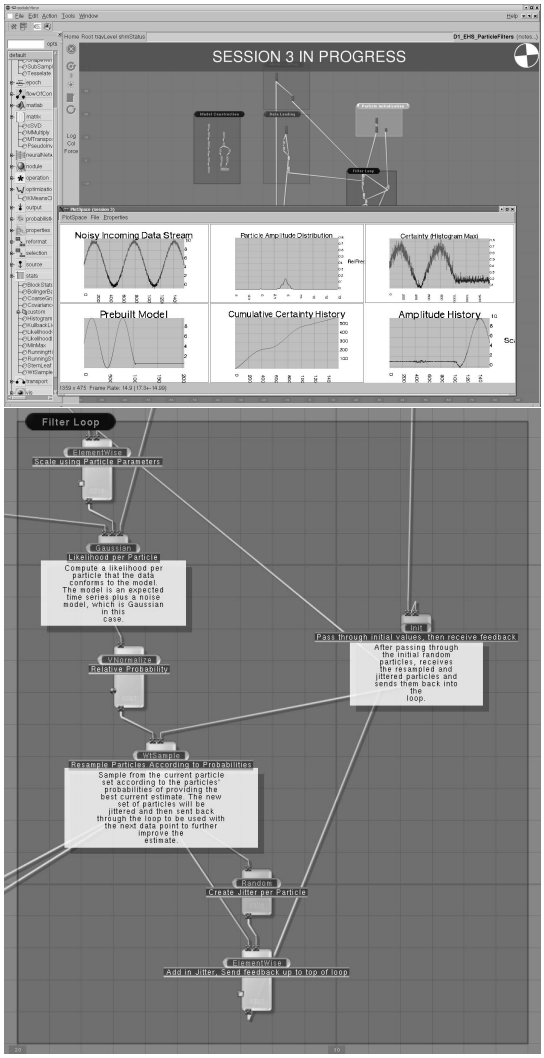

Figure 1: Top image portrays a particle filter system used for state tracking, the bottom image shows a magnified view of the filter loop which has feedback

3.1 Nodule Operation The functional units of a network are referred to as nodules. Each nodule is a separate executable responsible for generating, transforming, or visualizing data and/or parameter values. Typically, these are short (less than 200 lines of code) C++ executables, however anything that can communicate via our property system can be considered a nodule. For example, we have written several "mex" functions (C routines accessible in the Matlab scripting environment) that permit one or more Matlab instances to participate in a network session.

Nodule design focused on four functions most nodules would have to perform: parameter self-description, initialization, interprocess data communication and process shutdown. Self-description is necessary in a system where an external entity (the visual design environment) must discover new nodules at runtime and present them to a user for inte-

gration into an algorithm. Upon execution of the algorithm, nodule subsystems and parameters must be initialized. Nodules intercommunicate diverse forms and quantities of data, ranging from user-invoked events to infrequent threshold updates to fast-moving windows of data sampled at frequencies of multiple kilohertz. Nodules were defined to be separate processes: each started from a Unix shell environment with command-line parameters, each with access to shared memory and Unix ports, and each controllable by Unix signals.

Initial values, user invoked events, and threshold updates are handled by the property system, high bandwidth data communication is handled by shared memory, and process control is handled by Unix signal handlers.

3.1.1 Properties Each nodule maintains its own property tree, which can be thought of as a file system for parameter values. Each property is declared with a data type, a unique path, and a default value. Currently supported data types include the standard C base types, their multi-word (64+ bit) counterparts, and a string type. A property may also have an associated range, a description, as well as some hint flags which can used by the visual environment to choose an appropriate editing widget.

A property is used in much the same way as a normal C variable would be used (via C++ operator overloading), but it provides transparent capabilities for an external entity to set and interact with nodule parameters. When a nodule is run in "query" mode, its property tree is created with default values and then written to an XML file. The information in this file is used by the visual programming environment to determine how that nodule should appear in the workspace, create appropriate widgets to edit parameter values, and to create a high-level description of the nodule for on-line help.

The initial value of any property can be overridden at the command line by specifying the property's path and the new value. A nodule's properties can be interactively manipulated during runtime by other nodules. The mechanism by which this is accomplished is designed for low-bandwidth, infrequent communications. Each nodule maintains a lightweight server socket which it checks, at the nodule-writer's discretion, for requests for property values. Once a connection is established, changes to property values are immediately forwarded over the connection to the requesting nodule.

3.1.2 Shared Memory High-bandwidth data communication takes place primarily through shared memory, which is accessed using a platform-independent API that wraps the operating system-specific details. The primary design criteria for the API were high performance and strict interprocess locking. At the expense of explicit distribution of data across machines, the new system achieves practical throughput of as much as 200MB/s per processor with modern (circa 2002) multiprocessor PCs running Linux.

Because of the strict locking mechanisms, deadlocks are possible in certain algorithms. For example, given a data source, a model construction nodule, and a model evaluation nodule, imagine the following scenario:

The data source feeds both of the other nodules. The construction nodule needs to collect 1000 samples before it will output an initial model. The evaluation nodule needs a model before it can read from the data stream. The queue between the data source and the other two nodules only holds 500 samples at a time. The construction nodule will read all 500 samples and wait for more. The evaluation nodule will not read any of the samples, as it is waiting for a model. The data source cannot write another sample until the evaluation nodule has marked the data stream read. The system halts. We have allowed for the user to create these deadlocks to provide the maximum flexibility. To alleviate the deadlock potential, visual debugging tools have been added to the environment to show where data is queued and for what memory each nodule is waiting. In addition, queuing, loose locking and other flow-of-control nodules have been added to help solve such problems.

3.1.3 Signal Handling Nodule shutdown is triggered by a Unix "interrupt" signal. The signal is caught by signal-handling functions (which were initialized in the subsystem initialization), and a flag is set marking the process done. System calls that were waiting for a state change, such as shared memory locking and unlocking, are interrupted and return an appropriate error code. Most nodules run in a tight loop accessing shared memory, performing their algorithm and then checking for completion. Upon completion, the shared memory and locking resources are deleted. The signals are delivered to all of the nodules associated with an algorithm by the visual environment upon user cancellation. Nodules also detect when no more data will be written to the shared memories they are reading from, and if no one will be reading from the shared memories they are writing to. In this way, it is possible for an algorithm to complete: all nodules exit when all of the data has been processed.

3.2 Visual Programming Environment Operation The visual programming environment provides a list of available nodules, which is generated from the nodule property self description files explained above. The list is organized into a hierarchical tree and categories are color encoded to provide differentiation. This collection of available nodules is referred to as a "personality" and different personalities may be loaded, specific to the kind of problem a user wants to solve. The user can browse through nodule descriptions by highlighting them in the personality tree, which brings up an HTML document generated from the self description file. These documents list the nodule's properties and data types,

as well as any documentation the author has provided.

Once a user has found an appropriate nodule, it is dragged-and-dropped into the infinite 2-D workspace. At this point, it becomes a node in the network. This node may be connected to other nodes, and the user may right click on it to bring up a dialog box to edit the nodule's properties. One simple, but very powerful, feature is the ability to add notes to each node. By adding a few comments at each step of the algorithm, a new user can follow the data flow and quickly understand how the algorithm works.

Connections are made by dragging from an Output to an Input (for parameters), or from a Write to a Read (for shared memory). The connections are currently strictly type checked to avoid invalid connections. Each output of a shared memory (Outputs) and of a property (Writes) is allowed to have multiple connections. Each input to a shared memory (Inputs) and to a property is restricted to one connection.

3.3 Script Generation Once a wiring diagram has been produced, it has the form of a tree with cycles (because loops are allowed). This tree is then traversed and a script is generated. Each of the nodules in the network are standalone executables initiated by the generated script with the specified initial parameter values. The script is a standard Unix bash script. There are no timing issues of which block to run first because blocks communicate with each other via shared memory with protective locking mechanisms. Thus we allow the operating system to determine how the work load should be processed by having each nodule be a separate process. The only thing that all nodules have in common is a central logging facility which allows the user to see what is happening and is also used when shutting everything down. It is important to note that this central logger is not a bottleneck since the data flow between processes will be much much greater than the limited information communicated to the logger.

4 Applications

We focus on the progression of three research tasks performed by scientists using the framework. The first task involved examining spectra measured by a solar spectral flux radiometer (SSFR) which was mounted in a plane looking both up and down at light passing through the atmosphere. The second application was developing a framework network which would interpret Electromyographic (EMG) signals as gestures used for computer command. The final application was the progressive development of an Electroencephalogram (EEG) pattern recognition system for use in a brain computer interface (BCI).

4.1 Solar Spectral Flux Radiometer The solar spectral flux radiometer (SSFR) was designed to measure spectra from 300nm to 2000nm in 384 discrete bands. This resulted in 384 digital numbers representing the strength of spectra associated with each band during an interval of time. Samples were collected for each band once per minute. A typical data set consisted of several days worth of data which resulted in a matrix of 6872 by 621 (the scientists interpolate the 384 bands up to 621). The purpose of this data collection experiment was to obtain data sufficient to explain variations in the solar radiative energy budget. These variations are important to understand in order to know how changes in atmospheric composition are affecting the energy absorbed by the Earth's atmosphere. This in turn will affect climatic forecasts predictions of global warming.

The SSFR work first focused on identifying the major sources of variance in the spectral profiles. In order to understand which components were responsible for the majority of variance, the scientists performed a Principal Component Analysis (PCA) before our framework was ready to be used [1]. The PCA showed several major components, the first spectral profile related to the variation in cloud cover (liquid water vapor) (an intuitive result), and the other profiles corresponded to Rayleigh scattering, Chappius Ozone, and oxygen. The remaining variance was not identified. This approach was cause for concern for several reasons:

- We already know that clouds cause variations in the transmitted solar energy.
- Our understanding of changes in atmospheric species (e.g. CO2) and the resultant global climatic changes are very sensitive to absolute values, a 5% change in CO2 would result in significant changes to climatic forecasts.
- The variation is not the thing we really need to know.

This last item is the most significant because it shows the progression of thinking based upon the exploratory analysis of the data. Initially, the scientists thought that we should try to discover the spectral profiles which are most contributing to the overall variance. However, the majority of variance is composed of well known species and the remaining variance is actually the most interesting and impossible to identify using PCA. This led us to change the question that we were trying to answer, which in turn led to changing the data analysis method.

We are really interested in knowing the quantity of individual species in the atmosphere which then allows for us to directly calculate the affects on radiative transfer. This type of analysis has three requirements:

1. The data is representative of the phenomenon we wish to study
2. Ideally the measurements are absolute and not relative
3. We can formulate an accurate forward model of the system that we want to make inferences about.

The first point seems obvious, that if we want to infer the composition of the atmosphere we should be observing the atmosphere in some fashion. The more subtle point here is that the measurements are performed using a relative measurement instrument that was not designed to measure absolutes, however, the question we wish to answer really concerns absolutes. Moreover, the instrument itself may have unmodeled or unexpected errors which should be identified before incorrect inferences are made. In the SSFR data collection, the instrument suffered from a saturation problem during one day of data collection in one narrow spectral band. Given the quantity of data collected, this error was not detectable through visual inspection. However, before asking the science question, we used the framework to answer the following question: Is there anything in the data which seems to be statistically independent from the rest of the data set? Another way to say this: "Is this data representative of the underlying atmospheric process or are there other processes represented?"

To answer this we used the framework to perform Independent Component Analysis (ICA) requesting that only two components be formed. The form of ICA that we used was a Matlab version of FastICA [7]. The results of this are shown in Figure 2. There is a very dominant peak at the band that saturated in the first component. The second component represents the overall trend in the rest of the data. This leads us to believe that the first step in data analysis should be to answer the following question:
"Is this data representative of the process that I'm interested in or are there artifacts that need to be understood?"

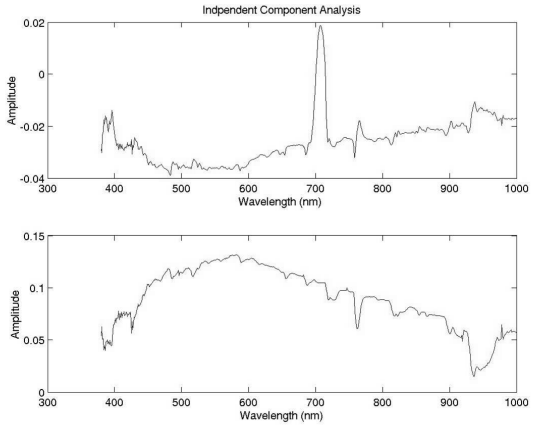

Figure 2: Independent component analysis results, top plot shows saturation anomaly, bottom plot represents average spectral profile

The network that produced both the PCA and ICA is depicted in Figure 3. This network demonstrates our interface to algorithms which have already been developed for Matlab.

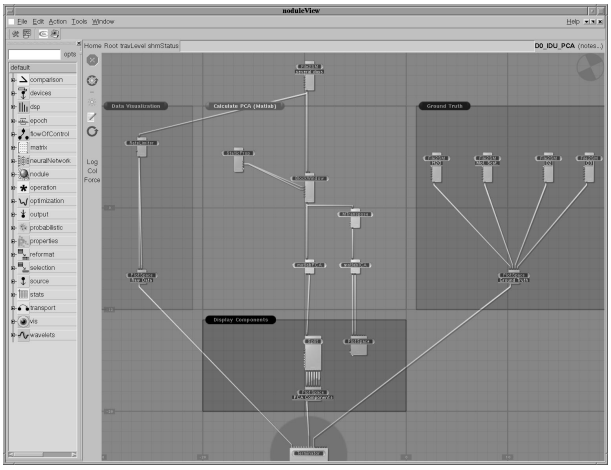

Figure 3: Network used to perform PCA and ICA via Matlab

In this case, we already had PCA and ICA Matlab code available and we chose to use our own mechanisms for reading in the data and producing plots.

After having resolved the saturation issue, we felt confident to ask the next question: what atmospheric constituents are affecting the spectrum that we are observing? Note that this is a significantly different question from the typical blind search commonly presented in the KDD literature. This approach requires that we formulate a hypothesis as to how light energy interacts with elements in the atmosphere. Thus we chose to formulate a new forward model of the atmosphere that allows us to embed this as the likelihood in a Bayesian formulation. When complete, we will be able to infer the atmospheric composition from the observed spectra by calculating the posterior distribution.

We are currently in the process of implementing the Bayesian method using this forward likelihood model to perform the parameterization. The derivation and development of this method are very time consuming and so we needed to perform the first steps of initial analysis to lead us to conclude that the more expensive steps were worth pursuing. Other steps in the algorithm development process which have to be performed using SPEAD include importing the data, modifying formats, and designing the data presentation that best aids the scientists in hypothesis formation.

Before using the collected data and our forward model to make inferences concerning the atmosphere, we have started developing an instrument model for improving the calibration process. The parameters of this instrument model are updated as data is collected. The following summarizes the lessons we have learned from this application:

- Ask the right question and then consider the needed tools/models to answer this question.
- Model noise and anomalies to prevent inappropriate

conclusions

- Data analysis should take into account each step in the data collection process.

4.2 Electromyographic Gesture Recognition EMG based gesture recognition has significantly different data processing requirements from the SSFR data application. The EMG data that was collected for gesture recognition typically consisted of 8 to 16 channels collected at 2000 Hz [2]. The first question that we wanted to answer was "Is it possible to use EMG for controlling a computer simulation?" This question was too broad to be answered immediately and therefore had to be broken down into the following categories:

- Where should electrodes be located to differentiate gestures?
- Given a set of gestures, what is the natural variability for nominal behavior?
- What transformation should be done to the data to best distinguish between gestures?
- What pattern recognition method for data streams provides the best accuracy? Is it possible for a computer to recognize EMG based gestures in an everyday setting?
- How much lag is acceptable to a user gesturing to control a computer?

These questions relate more to experiment formulation and engineering than to addressing an inferential science question. However, there are important science questions which can be intermixed within this process such as: "How do muscles in the forearm coordinate in order to produce a keystroke as observed by surface EMG?"

The electrode position question can be answered using information theoretic practices. For example, we could repose this question in a couple of different ways:

1. Which channels are most independent for a particular gesture?
2. In comparing gestures 'one' and 'two', which channels show the greatest difference?

In formulating the problem this way, we are able to make use of ICA to answer question one, and the Kullback-Leibler distance measure for question two. The network to determine the statistical difference between four gestures is shown in Figure 4. The network operates on 11 channels of surface-sensed forearm EMG, reading data streams containing multiple instances of finger presses. These "events" are further broken into time subsections since the acceleration, constant force and release sections of each event differ statistically

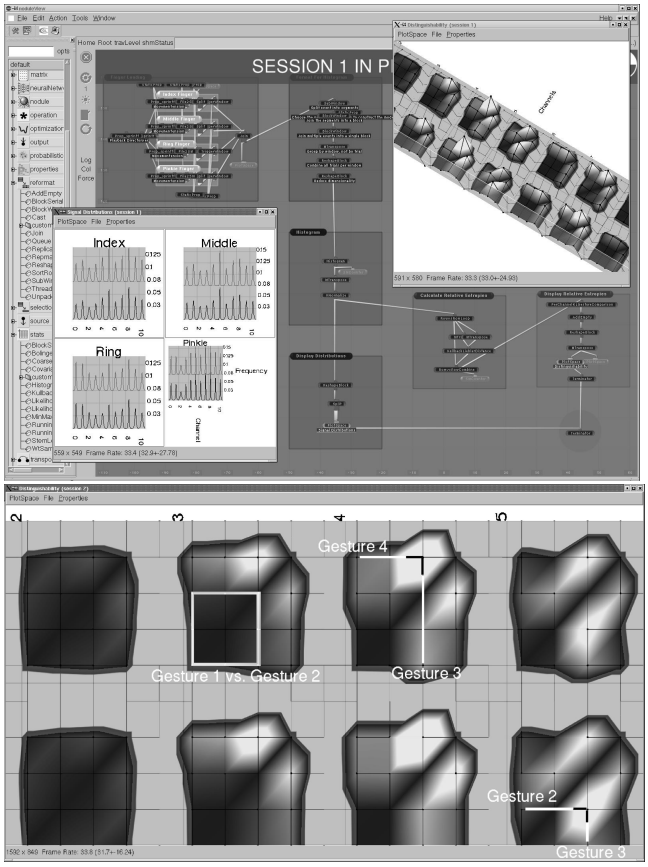

Figure 4: Top: A network used to determine the statistical distance between four gestures using 11 differential channels of forearm EMG. Bottom: An expanded view of the Kullback-Leibler distances between gestures per channel. Each channel has a 4x4 matrix (row 1, column 1 in the lower left per channel) of differences comparing one of four gestures against the others.

from each other. The data is reformatted to allow the construction of one histogram per gesture per time subsection per channel. All of these histograms are compared against each other using KL distance, and a subset of the resulting matrices is selected: a 4x4 matrix per channel per time subsection showing the distance between each gesture. In the "statistical distance" plot in Figure 4, the 4x4 matrices for channel 4 show a particularly high distance between gestures 3 (third finger press) and 4 (fourth finger press). Similarly, channel 5 is good at making this distinction and better at distinguishing between gestures 3 and 2 (second finger press). Channel 2 does not appear to be very good at making any distinctions. Along the diagonal, the distances are 0, as these are comparisons of each gesture against itself. Also, none of these channels are particularly good at distinguishing between first and second finger presses.

The issue of how to transform the data and which method to select for recognizing the patterns depends entirely upon the context of the question being asked. There are ultimately two approaches: black box and model based, and naturally there are many gradations in between. The black box method has been very successful in being able to map inputs to outputs, such is the case with neural networks. If a question is answered using a black box approach, it is difficult to make intelligent inferences about the underlying problem, and is nearly always the case that we start asking questions concerning the method rather than the problem at hand. A fully modeled system requires a great deal of mathematical competence and time to assemble but ultimately may be the best for further inquiry. Often in proof-of-concept work we are in a rush to see if the idea is at all feasible. In the case of EMG pattern recognition it is possible to use a prototype such as a hidden Markov model in our framework [2]. If our goal is a feasibility study with no further inquiry required, then this is a relatively quick way to proceed. The next stage will be to have as fast a system as possible with as much accuracy as possible. Inevitably, questions will arise concerning accuracy that will necessitate having some intuitive model of the underlying system. Our framework has been used in both manners. In the blind manner we developed a gesture recognition system using HMMs [8] with a moving average feature space which worked well enough to provide a demonstrable system. However to further this work required that a model be proposed which could be verified via experiments. Here is the typical sequence of events (some occurring over several years):

- Pose a question exploring a possibility, try to quickly see what others have done and extrapolate.
- Preliminary success results in wanting to do best possible, this raises question what is the best possible?
- Model is formed from intuition gained on working with black box model, model is parameterized.
- Model evaluation and refinement iterates with new data and formulations.

This can be translated into a series of questions that can be answered using our framework for the EMG gesture recognition problem:

1. Is it possible for a HMM to be trained to recognize EMG based gestures?
2. What are the best features and HMM configurations to achieve the greatest accuracy?
3. What is the simplest physiological model that represents our observations? (muscle coordination)
4. Which muscles can we infer are contracting given our observations?

Note that although this started as an engineering issue it evolved to a state of asking an inferential question regarding muscle coordination. These four questions were each answered by using our framework. The first was answered by establishing a network which smoothed the data with a moving average and then fed the resulting transformation into a HMM. This proved to be successful as a proof-of-concept but did not yield clues as to what could be done to improve the system because the underlying physiology was not explicitly modeled. The second question concerning the best features and HMM configuration has in general been answered by many researchers via trial and error. However, a better understanding of the underlying physics (of the physiology) of the system to be modeled would help provide an answer. In our work we first started with the trial-and-error approach and are now progressing to a more physiological based model. This model is what is underlying the final question regarding muscle contraction and coordination. Although the same underlying model may be used across a population, each individual will have a different parameterization.

Note that the framework has allowed us to progress from static analysis of collected EMG data sets, to one which is used to perform the analysis on-line. This transition process was painless because we were able to use nearly the same wiring diagram by changing the source of the data from a file to the data acquisition (DAQ) card. This allows us to do live demos without a subject by streaming data from a file as if it were from a DAQ card. It also allows for data previously recorded to be modified to test the effects of different types of perturbations.

The lessons that we have learned from using our framework for this application include:

- Make the transition from batch to streaming analysis as painless as possible by planning ahead and providing the appropriate infrastructure
- Try to avoid black-box approaches if further inferences are to be made

4.3 Electroencephalogram Pattern Recognition In the Electroencephalogram (EEG) pattern recognition work [3] 64 channels of data were collected at 1000 Hz. The subject was asked to try to control the movement of a needle on a control gage. The basic question was: “Is it possible to obtain two degrees of freedom movement via pattern recognition of EEG signals?” This question is motivated by the desire to be able to perform two dimensional cursor control using thought, and would be the first step to having a functioning brain computer interface system. In the neuroscience

literature it is well documented that when a person is at rest, the EEG energy becomes strong in the mu-rhythm band (8–12 Hz.); when motion commences, this energy dissipates which is known as desynchronization. Thus, the first step in producing the cursor control system seemed to be taking the signal from the critical electrodes (in our case C3 and C4) and passing this through a filter at 8.7 Hz (designed in Matlab, the FIR filter was used in our filter nodule). A network was constructed to accomplish this so that a subject could see the results on a moving dial. An example of this feedback produced by the framework is shown in Figure 5. Then we trained subjects using this streaming network to try to move the dial. The results were mixed (one subject was very good at this, several others had difficulty). The data was further studied off-line to determine if perhaps a coarse grain entropy measure might be more effective. This coarse grained entropy algorithm was then substituted in place of the filter during a live data collection experiment, and then the subject was allowed to train while the threshold values were manipulated to improve training using our property communication mechanism. This method of dynamically trying different algorithms during a live session proved to be very helpful to both the neuroscientists and the subjects.

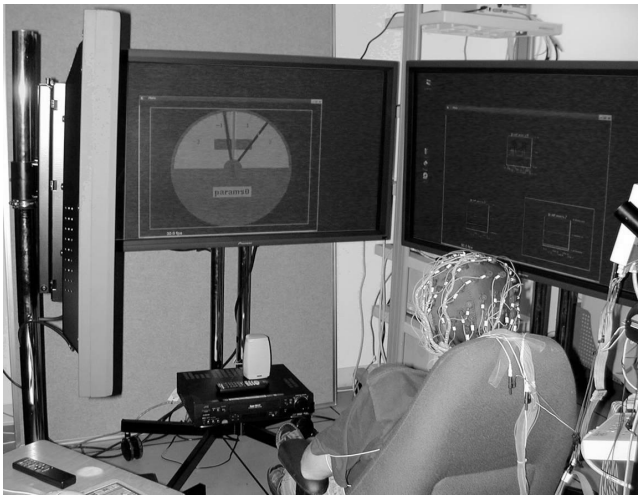

Figure 5: Brain computer interface training provided by framework which is processing EEG in real-time and presenting results on a dial

Our next step in this work is to use Support Vector Machine (SVM) classifiers to try to improve the accuracy of recognizing movement from no-movement. At first this involves processing data files in batch mode, and then we will switch over to using SVM during live recording/training sessions. So far the most important lesson that we have learned from this work is that it is very useful to support the ability to swap between different algorithms during a streaming data collection experiment to see which method works best with a particular participant.

5 Final Observations

We started this project with the idea of providing both Earth and life science domain experts with exploratory tools with which they would then make relevant discoveries. This 'over-the-fence' attitude is prevalent in the machine learning literature and yet is misguided. We are convinced that through the combination of exploration and domain-based model development, ground breaking questions can be posed and answered in an inferential study.

In the development of our framework we have learned several lessons:

- The flow of data should be easy to change without low-level programming
- The swapping of algorithms should be easily performed during streaming
- The data collection process should be modeled and the risk of making inferences from bad data must be qualified
- The science questions need to be posed before computational methods are considered

The first two items have been addressed through the capabilities of our framework:

- Easy to program graphical wiring diagrams.
- Fast 2-D and 3-D graphics for interaction during streaming.
- Multi-machine distributed computing support for a heterogeneous network.
- Capability to swap algorithms acting on the data stream.
- Support of existing infrastructure such as The Mathwork's Matlab and LAPACK.
- Graphical syntax checker to reduce burden of memorizing syntax.

It is our hope that use of the framework will facilitate the use of careful modeling and analysis of data and the associated collection process. It is dangerous to provide tools and think that this is the end solution; ultimately, only close collaborations provide the scientific end solution.

In the future we plan to make this software available to the community at large. Our current plans are to increase the number of available nodules, to complete the implementation of a Bayesian model for spectral atmospheric inference, and to integrate other packages.

References

[1] M. Rabbette and P. Pilewskie. Multivariate analysis of solar spectral irradiance measurements. *J. Geophys. Res.*, 106(D9):9685–9696, May 2001.

[2] Charles Jorgensen, Kevin Wheeler, and Slawomir Stepniewski. Bioelectric flight control of a 757 class high fidelity aircraft simulator. In *Proceedings of the World Automation Congress*, 2000.

[3] Leonard J. Trejo, Kevin R. Wheeler, Charles C. Jorgensen, Roman Rosipal, Sam Clanton, Bryan Matthews, Andrew D. Hibbs, Robert Matthews, and Michael Krupka. Multimodal neuroelectric interface development. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, in press*, 2003.

[4] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Proceedings of the First Biennial Conference on Innovative Database Systems (CIDR03)*, January 2003.

[5] Chandrika Kamath. Sapphire: Large-scale data mining and pattern recognition. Technical Report UCRL-TB-132076, Lawrence Livermore National Laboratory, January 1999.

[6] Jun S. Liu and Rong Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[7] A. Hyvrinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[8] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, February 1989.